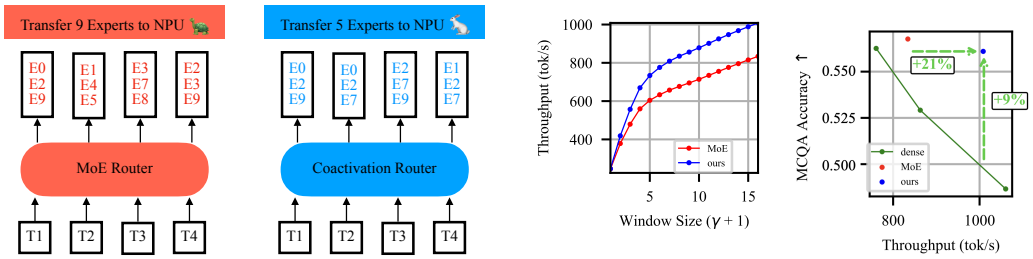
EFFICIENT MIXTURE-OF-EXPERTS WITH SPECULATIVE DECODING VIA EXPERT COACTIVATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models are increasingly deployed alongside Speculative Decoding (SD) to accelerate inference, but combining the two is challenging. SD improves the inference speed of dense models by verifying groups of tokens in parallel. However, the inference speedup for SD with MoEs depends heavily on the number of tokens being verified. Using more verification tokens results in more experts being transferred from DRAM to the Neural Processing Unit (NPU), which increases the memory transfer cost. This negatively impacts model runtime, as memory transfer is typically the bottleneck in inference. In this work, we investigate the impact of MoE router design during training on the speed of MoEs with SD. We find that routers with high degrees of expert coactivation result in much faster runtimes, mitigating the impact of using more verification tokens. Motivated by this observation, we assess the impact of various router design choices on expert coactivation and runtime using billion-parameter transformer models. We find that combining a global load-balancing loss, shared experts, a consistency loss, and an autoregressive expert selection mechanism during training results in significantly stronger expert coactivation. This increased coactivation translates into higher overall runtime throughput: our exploration yields a model that improves throughput by 21% over MoE baselines, while maintaining on-par accuracy with the baseline MoE.



(a) MoE (b) Our MoE (c) Throughput (d) Accuracy

Figure 1: An overview of our method. (a): Standard MoEs require transferring many experts to the Neural Processing Unit (NPU) during SD, creating a bottleneck. (b): Our MoE improves expert coactivation, which increases throughput. (c): Our method improves throughput across all SD verification window sizes. (d): Our method retains accuracy benefits of MoE while improving throughput.

1 INTRODUCTION

Mixture-of-Experts (MoE) models have demonstrated strong results in language-modeling tasks (Jacobs et al., 1991; Muennighoff et al., 2024; Shazeer et al., 2017; Zhou et al., 2022). Their appeal lies in their ability to scale up model sizes while reducing training and inference cost by activating only a subset of the model's feedforward network (FFN) parameters during the forward pass. This technique has become widely adopted among today's leading open-weight LLMs (DeepSeek-AI et al., 2025; Muennighoff et al., 2024; Yang et al., 2025; OpenAI, 2025; NVIDIA, 2026; Xiaomi LLM-Core Team, 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In a transformer, MoE and dense models differ primarily in the FFN, where an MoE dynamically routes each token to a subset of expert FFNs (see section 2 for a formal treatment). This lets MoE LLMs match the latency and FLOPs of a much smaller dense model while approaching the quality of a much larger one.

However, deploying MoEs for inference presents challenges. Efficient inference often relies on Speculative Decoding (SD) (Leviathan et al., 2022), which produces multiple tokens per step using a cheap approximation model to draft tokens that the target model verifies in parallel (see section 2 for details). Indeed, several of the newest frontier MoEs now ship native speculative decoding support built into the MoE architecture itself (NVIDIA, 2026; Xiaomi LLM-Core Team, 2026), underscoring the practical importance of the problem we tackle in this work.

When performing inference with an MoE, the verification cost is no longer fixed. In this case, a parallel decoding call will take significantly longer than a single-token decoding call. This is because neighboring tokens may choose different experts, which means that a greater number of expert weights need to be transferred from dynamic random access memory (DRAM) to the Neural Processing Unit, or NPU (e.g. a TPU, GPU, or other accelerator) compared to single-token inference. Since decoding is bottlenecked by model weight transfers during inference, this extra parameter transfer will result in a significant decrease in inference throughput. This will reduce the computational savings normally provided by using sparsely-activated MoEs. In the worst case, all N experts could be transferred to the NPU. Concurrent work has begun to characterize this interaction between SD and MoE routing from a serving-time perspective (Saxena et al., 2025; Huang et al., 2025); in this work, we instead ask whether the MoE itself can be trained to make SD verification cheaper.

Our goal is to improve *expert coactivation* – e.g. to encourage neighboring tokens to choose similar sets of experts during verification to recover the performance benefits of SD. Rather than intervening at serving time, we achieve this entirely by changing how the MoE is *trained*: our method is simple, and applies four changes to the standard MoE training setup, leaving the SD algorithm and the deployed drafting mechanism untouched. First, we replace the standard *local* load-balancing loss with a global one (Qiu et al., 2025). Second, we use shared experts (DeepSeek-AI et al., 2025). Third, we add a novel *consistency loss* to encourage neighboring token's router logits to be similar. Finally, we add a novel *autoregressive expert selection* mechanism to ensure expert selections are similar for neighboring tokens.

Figure 1 shows an overview of our approach. Figure 1a shows a standard MoE router verifying four tokens, requiring nine unique experts to be transferred from DRAM to the NPU; Figure 1b shows the same verification under our coactivation techniques, reducing this to five experts. Figure 1c shows that our method improves throughput across all SD window sizes γ. Figure 1d shows results for an MoE with 1 billion total and 353 million active parameters, compared to a 352 million parameter dense model: our method retains the accuracy benefits of MoEs (9% relative improvement over dense) while achieving 21% higher throughput than standard MoEs. Our contributions are as follows:

- We analyze the performance of MoEs with SD and show that expert transfers decrease throughput.
- We show that improving expert coactivation decreases memory transfer costs and improves SD's efficacy for MoEs.
- We propose a combination of training-time techniques – a global load-balancing loss, shared experts, a novel consistency loss, and a novel autoregressive expert selection – that improve expert coactivation during inference with SD, thereby improving overall throughput.

2 Background

We provide a brief overview of MoEs and of SD in the following sections.

2.1 Mixture-of-Experts

MoEs are a more compute-efficient alternative to dense transformers (Vaswani et al., 2017), applying structured sparsity dynamically at inference time.

Model	message drafting	message rewriting	tone adjustment	machine translation
Dense 352M	4.45×	2.54×	1.81×	3.50×
MoE X=16 K=4	3.38×	1.93×	1.42×	2.65×

Table 1: Throughput improvements provided by SD for a dense 352M parameter model, and an MoE with 353M active parameters and 1032M total parameters (with 16 experts and topk set to 4).

A standard MoE replaces the FFN with several *expert* FFNs; a *router* R computes per-expert weights for each token, giving an FFN output of

$$\text{FFN}_{\text{MoE}}(x) = \sum_{i=0}^{N-1} R(i)E_i(x), \tag{1}$$

where x is the input token, N is the number of experts, E_i is the i^{th} expert, and $R(i)$ is its router weight. R includes a TopK function ($R(i) > 0$ for k values of i), so $E_i(x)$ need only be computed for k unique experts.

2.2 Speculative Decoding

Speculative Decoding (SD) is a general technique for improving inference speed. It uses a smaller *approximation model* alongside the deployed *target model*: the approximation model autoregressively produces γ candidate tokens, and the target model verifies all $\gamma + 1$ tokens (the input plus the γ candidates) in a single parallel call, accepting or rejecting each candidate such that the output distribution exactly matches autoregressive decoding with the target model alone. We refer readers to Leviathan et al. (2022) for the full verification procedure.

The key insight leveraged by SD is that model inference for modest context lengths is bottlenecked by memory bandwidth between DRAM and the NPU. For this reason, a parallel decoding step with a modest window size (for instance, 8) takes approximately the same amount of time as a decoding step with a window size of 1, since the same number of model weights are transferred to the NPU in both cases. By speculating future tokens with a computationally inexpensive approximation model, SD allows a target model to produce multiple tokens per inference call, resulting in higher throughput. The improvement in throughput depends on the acceptance rate, or the probability of accepting tokens produced by the approximation model. The overall throughput can be approximated as

$$s = \frac{v(\gamma, \mathcal{A}, \mathcal{M})}{\mathcal{A}(1)\gamma + \mathcal{M}(\gamma + 1)}, \tag{2}$$

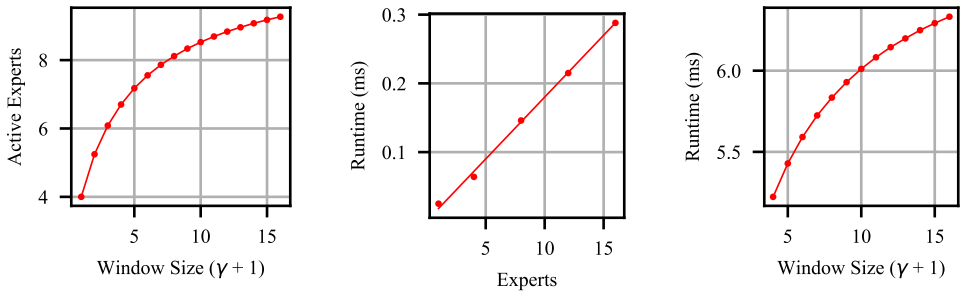
where $v(\cdot)$ is the (average) number of tokens produced in an SD step, $\mathcal{A}$/$\mathcal{M}$ are the approximation/target models, and $\mathcal{A}(x)$/$\mathcal{M}(x)$ are their runtimes on x input tokens. Note that $v(\cdot) \leq \gamma + 1$, since at most $\gamma + 1$ tokens can be produced in one step. Throughput is measured in tokens generated per second.

3 Motivation: Reduced Performance of SD on MoEs

Our goal is to improve the inference-time throughput of MoEs used with SD. SD greatly improves throughput for dense models. However, we observe that MoEs benefit less from SD. This is because the SD verification step for MoEs requires transferring more experts than single-token inference. In other words, the standard assumption made by SD (i.e. that parallel and single-token inference take the same amount of time) is violated. To illustrate this point, we make the following observations:

Observation 1: SD gives strong throughput increase to dense models, but lower throughput increase for MoEs. In Table 1, we view the improvements in throughput provided by SD for a dense model and an MoE for various tasks. Both models have the same number of active parameters, and the same single-token inference speed (240 tokens/s). However, the SD speedup for MoEs is reduced across the tasks. We investigate the source of this reduction in Observations 2 and 3.

Observation 2: Parallel decoding increases the DRAM-to-NPU communication cost for MoE. During parallel decoding, the number of experts activated within a layer of the MoE increases as the



(a) Experts vs Window Size (b) Runtime vs Experts (c) Throughput vs Window Size

Figure 2: Larger window sizes increase the runtime of MoE FFN layers. (a): Larger window sizes increase the number of active experts. (b): Increasing the number of active experts increases the runtime. (c): The impact of window size on runtime.

window size increases. In Figure 2a, we show the average number of active experts (per-layer) as a function of the SD verification window size (i.e. the number of tokens decoded in parallel by the target model). As the window size increases, the number of experts increases. This corresponds to an increase in the number of parameters that need to be sent to the NPU for computation.

Observation 3: This increase in DRAM-to-NPU communication slows down the model. The increase in the number of experts in Observation 2 increases inference cost. In Figure 2b, we benchmark the runtime of an MoE FFN layer as a function of the number of active experts. The trend is linear, with a non-zero y-intercept. As the number of experts increases, the runtime increases.

Summary of Observations 1-3: Reduced performance of SD with MoEs is the result of increasing DRAM-to-NPU communication as window sizes increase. In Figure 2c, we visualize the runtime as a function of SD window size. For MoEs, the runtime increases steadily as window size increases. This reduces the speedup provided by SD, especially at high window sizes.

4 Improving Expert Coactivation To Improve Speed

To improve the runtime of MoEs with SD, we can reduce the number of experts activated by the set of tokens in the SD window. We hypothesize that we can encourage neighboring tokens to work together to choose overlapping experts with a minimal impact on accuracy. We accomplish this with four training-time changes to the MoE router: a global load-balancing loss (subsection 4.1), shared experts (subsection 4.2), a novel consistency loss (subsection 4.3), and a novel autoregressive expert selection (subsection 4.4). Finally, we demonstrate that these methods increase expert coactivation in subsection 5.2.

4.1 Global Load-Balancing Loss

MoEs typically use a load-balancing loss to ensure that all experts are used by the network (Shazeer et al., 2017; Muennighoff et al., 2024). This helps prevent experts from becoming underutilized – if an expert has a very low probability of being selected, the load balancing loss can help it recover.

The load balancing loss computed over a set of neighboring tokens τ is computed as

$$\mathcal{L}_{Bal}(\tau) = N \sum_{i=0}^{N-1} f(i,\tau)p(i,\tau), \tag{3}$$

where N is the number of experts, $f(i,\tau)$ is the fraction of tokens in τ that selected expert i, and p is the total routing probability among tokens in τ routed to expert i.

On training clusters, processes across multiple nodes compute gradient updates in parallel, with the global batch $\mathcal{B}$ divided into micro-batches per process. Usually, the load-balancing loss is computed

separately per micro-batch and averaged, to avoid inter-node data transfer (Qiu et al., 2025). The full loss across $\mathcal{B}$ is thus the average of local losses over each micro-batch of τ tokens,

$$\mathcal{L}_{Local}(\mathcal{B}) = \frac{|\tau|}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \mathcal{L}_{Bal}(\tau) \tag{4}$$

$$= \frac{|\tau|}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} N \sum_{i=0}^{N-1} f(i, \tau) p(i, \tau). \tag{5}$$

One drawback of this local loss is that it encourages the router to select all experts per micro-batch, *even if the micro-batch's data come from the same domain*, adversely impacting expert specialization (Qiu et al., 2025). To alleviate this, we use a global load-balancing loss instead, formulated as

$$\mathcal{L}_{Global}(\mathcal{B}) = \mathcal{L}_{Bal}(\mathcal{B}) \tag{6}$$

$$= N \sum_{i=0}^{N-1} f(i, \mathcal{B}) p(i, \mathcal{B}). \tag{7}$$

This requires more inter-node communication than using local loss. However, we find in practice this step does not cause a bottleneck in training.

4.2 Shared Experts

One simple way to improve expert coactivation is to implement shared experts (DeepSeek-AI et al., 2025). In this formulation, shared experts are activated by all tokens, whereas routed experts are selected per token via the router's top-k mechanism. Given N total experts, S of which are shared, the MoE's FFN output can be computed as

$$\text{FFN}_{\text{MoE-Shared}}(x) = \sum_{i=0}^{S-1} E_i(x) + \sum_{i=S}^{N-1} R(i) E_i(x). \tag{8}$$

This guarantees that at least S experts will be shared among all tokens.

4.3 Consistency Loss

We introduce a novel *consistency loss* to encourage neighboring tokens to produce similar router logits. Specifically, we add a KL divergence loss between the router logits for token i and $i+1$:

$$\mathcal{L}_C = KL(\text{SG}(R(i)) || R(i+1)),$$

where $R(i)$ denotes the router logits for the i^{th} token, KL denotes the Kullback-Leibler divergence (Kullback & Leibler, 1951), and SG denotes the stop-gradient operator. The consistency loss is multiplied by a scale factor λ_C.

4.4 Autoregressive Expert Selection

Finally, we introduce a novel autoregressive expert selection algorithm to replace the standard per-token TopK selection in the router R, under which neighboring tokens select experts independently. Within each window of $\gamma + 1$ tokens, we autoregressively choose experts for each token so that neighboring tokens' selections become correlated: each subsequent token retains at least $k - l$ experts chosen by the previous token, then fills the remaining selections with its own highest-ranked experts.

More formally, let E_i denote the set of experts chosen by token i. Let $\text{top}(k, X, \tilde{R})$ denote the highest-ranked k elements from the set X using the corresponding router weights $\tilde{R}$ (i.e., the router

Model	layers	model dim	FFN dim	Active Params (M)	Total Params (M)
Dense 352M	24	1024	3072	352	352
Dense 522M	24	1024	5376	522	522
Dense 692M	24	1024	7680	692	692
Dense 862M	24	1024	9984	862	862
Dense 1032M	24	1024	12288	1032	1032
MoE X=16 K=2	24	1024	12288	240	1032
MoE X=16 K=4	24	1024	12288	353	1032
MoE X=16 K=6	24	1024	12288	466	1032

Table 2: A summary of models used in experiments.

weights prior to top-k selection). Further, let $[N]$ denote the index set of all experts, $\{0, 1, ..., N-1\}$. Within an SD window of $\gamma + 1$ tokens, we compute the selected experts for each token as

$$E_i = \begin{cases} \text{top}(k, [N], \tilde{R}(i)) \text{ if } i = 0 \\ \alpha_i \cup \text{top}(l, [N] \setminus \alpha_i, \tilde{R}(i)) \text{ if } i > 0, \end{cases} \tag{9}$$

where $\alpha_i = \text{top}(k - l, E_{i-1}, \tilde{R}(i))$ represents the top $k - l$ experts chosen by the *previous* token, ranked by the *current* token's router weights $\tilde{R}(i)$.

By construction, we retain at least $k - l$ experts from the previous token, and add at most l new experts when selecting experts for the current token. Thus, for a window size of $\gamma + 1$ tokens, the number of experts processed is at most $k + \gamma l$.

5 Results

We demonstrate in subsection 5.2 that our methods improve throughput, and here verify that they do not harm accuracy.

5.1 Experimental Details

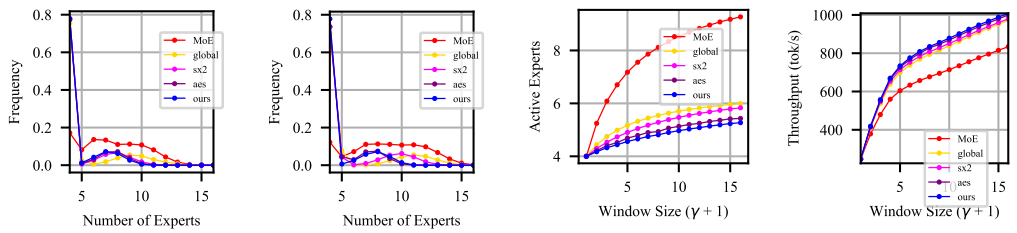
A summary of our transformer-based LLMs appears in Table 2. Our smallest dense model has 352M parameters, following Gunter et al. (2024); other dense variants scale up the FFN dimension, and our MoEs are obtained by converting that FFN to an MoE layer at each scale. Our approximation model used in SD is 3% of the target MoE's size, incurring negligible overhead.

For the fairest possible comparison, we train all models ourselves from scratch. All hyperparameters including learning rate, batch size, number of iterations, loss function settings, etc. are identical for MoE and dense models. MoE training additionally includes a load-balancing loss and a router z-loss (Zoph et al., 2022), which have no dense-model counterpart. See Appendix B for hyperparameter settings. Our models are trained on large-scale text data for 6 trillion tokens, the point at which the loss saturates.

5.2 Our Methods Improve Expert Coactivation

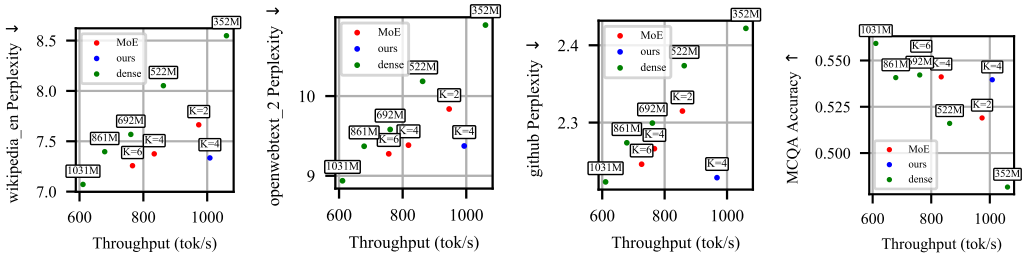
We investigate the improvements in expert coactivation using our four methods in Figure 3. We ablate global load-balancing, shared experts, and autoregressive expert selection individually, since these directly control expert selection; our consistency loss is instead auxiliary, encouraging overlap only in the combined method (**ours**, in blue). For autoregressive expert selection, we choose $l = 1$. Note that all methods use global load-balancing except the **MoE** baseline (in red), since local load-balancing harms expert specialization (Qiu et al., 2025). Our model has $N = 16$ experts and $k = 4$.

In Figure 3a, we plot the total number of coactivated experts (x-axis) and the frequency (y-axis) for layer eight of the model when processing an eight-token window of the Wikipedia (Gao et al., 2020) dataset. Switching from a local loss (**MoE**, red) to a global loss (**global**, yellow) drastically increases



(a) Layer 8 (b) Layer 16 (c) Active Experts (d) Throughput

Figure 3: Expert coactivation analysis for an MoE baseline with local load-balancing (**MoE**), global load-balancing (**global**), 2 shared experts (**sx2**), autoregressive expert selection (**aes**), and all of the above with our auxiliary consistency loss (**ours**). (a) The frequency of active experts in an 8-token window at layer 8. (b) The frequency of active experts in an 8-token window at layer 16. (c) The average number of active experts as a function of window size. (d) Our improvements in coactivation translate to improvements in throughput at all SD window sizes.



(a) (b) (c) (d)

Figure 4: Perplexity (or accuracy) vs. throughput, comparing dense (green), MoE (red), and our MoE (blue), for (a) Wikipedia, (b) OpenWebText2, (c) Github, and (d) Question Answering. Our MoE produces the strongest results.

the frequency at which only four experts are selected across the eight-token window. We analyze layer sixteen of the model in Figure 3b and see a similar pattern.

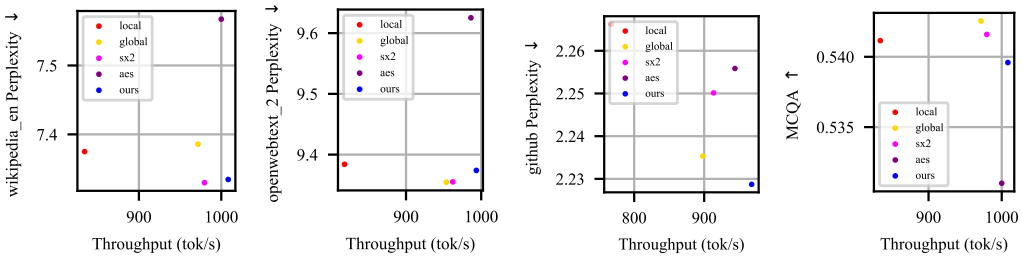
In Figure 3c, we analyze the average number of active experts across a variety of window sizes. By combining all methods (**ours**, blue), we achieve the lowest average number of active experts. In Figure 3d, we observe that this translates to the strongest throughput at all window sizes.

5.3 Our Method Outperforms Dense and MoE Baselines

Here, we demonstrate that our method improves throughput over MoE baselines without adversely impacting accuracy. In Figure 4 we plot the throughput-accuracy trade-off curve of our method compared to dense and MoE baselines. We investigate the perplexity on Wikipedia (Gao et al., 2020), OpenWebText2 (Gao et al., 2020), and GitHub (Gao et al., 2020). We also investigate the accuracy of question-answering on a combination of Arc Easy (Clark et al., 2018), Arc Challenge (Clark et al., 2018), Hellaswag (Zellers et al., 2019), Lambada (Paperno et al., 2016), PiQA (Bisk et al., 2019), SciQ (Welbl et al., 2017), WinoGrande (Sakaguchi et al., 2019). In all cases, our model achieves a pareto-optimal throughput-accuracy trade-off. Our model is trained with $k = 4$, and matches the accuracy of our MoE baseline with $k = 4$; as shown in Figure 4d, our MoE achieves this on-par accuracy while improving throughput by 21% over the MoE baseline, due to increased coactivation. Our throughput surpasses that of an MoE baseline with only $k = 2$ experts chosen.

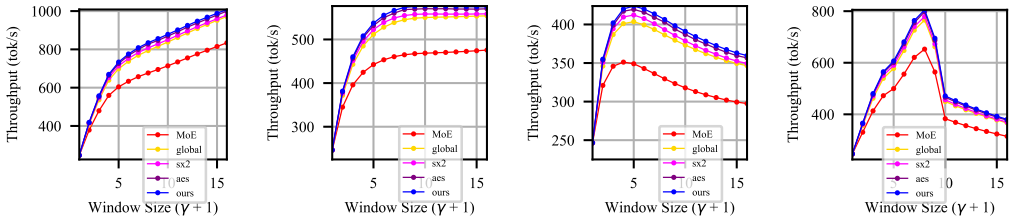
5.4 Ablation: Combining All Four Methods Gives the Best Performance

We ablate the throughput-accuracy trade-off provided by global load-balancing, shared experts, and autoregressive expert selection individually, versus combining them with our auxiliary consistency



(a) (b) (c) (d)

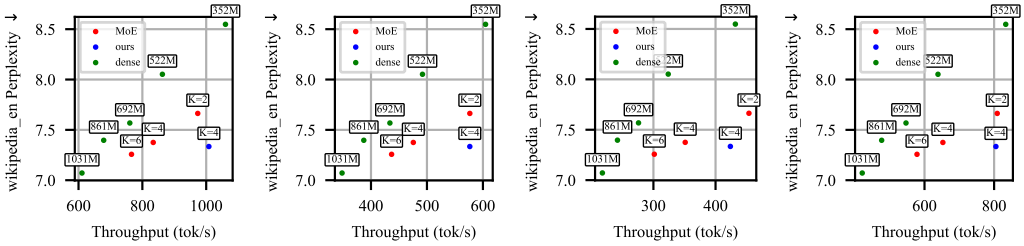
Figure 5: Perplexity (or accuracy) vs. throughput, ablating local load-balancing (**local**), global load-balancing (**global**), 2 shared experts (**sx2**), autoregressive expert selection (**aes**), and combining global, sx2, and aes with our auxiliary consistency loss (**ours**), for (a) Wikipedia, (b) OpenWebText2, (c) Github, and (d) Question Answering.



(a) Message Drafting (b) Message Rewriting (c) Tone Adjustment (d) Machine Translation

Figure 6: Throughput vs. SD window size for 4 different tasks.

loss (**ours**). In Figure 5d, we compare our combined method (**ours**, blue) with applying each method separately. Autoregressive expert selection (**aes**, purple) yields a small reduction in quality, but performance recovers when combined with our other methods (**ours**, blue).



(a) Message Drafting (b) Message Rewriting (c) Tone Adjustment (d) Machine Translation

Figure 7: Perplexity vs. throughput for acceptance rates corresponding to 4 different tasks.

5.5 Impact of Task Difficulty

Some tasks yield a higher SD acceptance rate than others: as seen in Table 1, *message drafting* is easier for SD, whereas *tone adjustment* is harder. We verify that our improvements translate across tasks. In Figure 6, we plot throughput vs. window size using SD acceptance rates for four tasks: our method has the highest peak throughput across all tasks, peaking at higher window sizes than baselines since coactivation is stronger.

For easier tasks (e.g., *message drafting*), our method and baselines peak at higher window sizes, since larger windows are more likely to have more tokens accepted; for harder tasks, both peak at lower window sizes. In all cases, our model improves peak throughput over baselines.

In Figure 7, we plot the throughput-accuracy trade-off curve of our method compared to dense and MoE baselines using SD acceptance rates for these tasks. Across all tasks, our method achieves a stronger trade-off. For easier tasks (e.g. *message drafting*), our method's advantage is stronger.

6 Related Work

MoEs scale model capacity while controlling inference cost by activating a sparse subset of expert FFNs per token (Jacobs et al., 1991; Shazeer et al., 2017; Zhou et al., 2022). Krajewski et al. (2024) and Ludziejewski et al. (2025) derive MoE scaling laws showing the compute-optimal expert count stays low until model scale exceeds several billion parameters. Our setting differs: we assume enough memory for all experts, with DRAM-to-NPU *bandwidth* as the binding constraint, not compute. Local vs. global load-balancing losses also affect expert specialization (Qiu et al., 2025), which we build on in subsection 4.1. Skliar et al. (2024) explore post-training cache-reuse techniques for loading experts from flash on mobile devices; we instead target the number of *distinct* experts requested per verification step. Instruction-conditioned pruning (Hou et al., 2025) offers an MoE-free route to sparsity, without retaining a single model across tasks.

Speculative Decoding (Leviathan et al., 2022) was originally analyzed for dense models, where verification is free regardless of window size. Two recent studies show how this breaks down for MoEs: Saxena et al. (2025) show speculation can inflate expert-weight transfers by 2-3$\times$, sometimes causing a net slowdown, and propose an adaptive policy; Huang et al. (2025) instead find MoEs can benefit *more* from SD under the right batch-size regime. Other systems work reduces verification-time expert loads: training-free per-layer expert budgeting (McDanel et al., 2026), cost-aware draft selection that avoids "expert scattering" (Xie et al., 2026), and commitment-weighted, self-sizing verifier expert sets (Liang et al., 2026). All of these act at inference time on a fixed, trained MoE; our approach instead modifies router training, and is complementary to all of them.

Closest to our approach, Suram (2026) train small ($\leq$340M-parameter) MoE routers for cache locality via auxiliary losses, but report a negative result: cache-miss reductions violate a pre-registered perplexity gate. Wu et al. (2026) instead adapt the MoE backbone with an auxiliary cache-prediction module, improving cache hit rates without retraining router selection. We differ in objective and scale: we target *coactivation within an SD verification window* rather than cache locality generally, evaluating at the billion-parameter scale, above where Suram (2026) observe the tradeoff.

A separate line of work trains the MoE itself to draft for itself, e.g. via self-distillation of a per-layer draft expert (Han, 2026) or self-speculative decoding on-device (Huang et al., 2026). These methods modify SD's drafting side, whereas ours modifies only the target model's router, and could be combined with any of them.

7 Conclusion

We investigate the performance of MoEs with SD. We find that SD demonstrates reduced efficacy on MoEs due to increased DRAM-to-NPU bandwidth usage during verification. We show that applying global load-balancing, shared experts, autoregressive expert selection, and consistency loss during MoE training encourages expert coactivation, improving throughput on downstream tasks without harming accuracy. These improvements hold across a variety of tasks and datasets.

AI use statement

We used generative AI tools for language editing: correcting grammar and reworking wordy paragraphs into more compact prose. We did not use generative AI to conduct research, generate research ideas, or draft any section of the manuscript from scratch; all text originates from the authors. We have reviewed all AI-assisted edits and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. *arXiv preprint arXiv:1911.11641*, 2019. URL

https://arxiv.org/abs/1911.11641. AAAI 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, Deepak Gopinath, Dian Ang Yap, Dong Yin, Feng Nan, Floris Weers, Guoli Yin, Haoshuo Huang, Jianyu Wang, Jiarui Lu, John Peebles, Kewei Ye, Mark Lee, Nan Du, Qibin Chen, Quentin Keunebroek, Sam Wiseman, Syd Evans, Tao Lei, Vivek Rathod, Xiang Kong, Xianzhi Du, Yanghao Li, Yongqiang Wang, Yuan Gao, Zaid Ahmed, Zhaoyang Xu, Zhiyun Lu, Al Rashid, Albin Madappally Jose, Alec Doane, Alfredo Bencomo, Allison Vanderby, Andrew Hansen, Ankur Jain, Anupama Mann Anupama, Areeba Kamal, Bugu Wu, Carolina Brum, Charlie Maalouf, Chinguun Erdenebileg, Chris Dulhanty, Dominik Moritz, Doug Kang, Eduardo Jimenez, Evan Ladd, Fang Shi, Felix Bai, Frank Chu, Fred Hohman, Hadas Kotek, Hannah Gillis Coleman, Jane Li, Jeffrey P. Bigham, Jeffery Cao, Jeff Lai, Jessica Cheung, Jiulong Shan, Joe Zhou, John Li, Jun Qin, Karanjeet Singh, Karla Vega, Kelvin Zou, Laura Heckman, Lauren Gardiner, Margit Bowler, Maria Cordell, Meng Cao, Nicole Hay, Nilesh Shahdadpuri, Otto Godwin, Pranay Dighe, Pushyami Rachapudi, Ramsey Tantawi, Roman Frigg, Sam Davarnia, Sanskruti Shah, Saptarshi Guha, Sasha Sirovica, Shen Ma, Shuang Ma, Simon Wang, Sulgi Kim, Suma Jayaram, Vaishaal Shankar, Varsha Paidi, Vivek Kumar, Xin Wang, Xin Zheng, Walker Cheng, Yael Shrager, Yang Ye, Yasu Tanaka, Yihao Guo, Yunsong Meng, Zhaoping Luo, Ouyang Zhi, Alp Aygar, Alvin Wan, Andrew D. Walkingshaw, Tzu-Hsiang Lin, Arsalan Farooq, Brent Ramerth, Colorado Reed, Chris Bartels, Chris Chaney, David Riazati, Eric Liang Yang, Erin Feldman, Gabriel Hochstrasser, Guillaume Seguin, Irina Belousova, Joris Pelemans, Karen Yang, Keivan A. Vahid, Liangliang Cao, Mahyar Najibi, Marco Zuliani, Max Horton, Minsik Cho, Nikhil Bhendawade, Patrick Dong, Piotr Maj, Pulkit Agrawal,

Qi Shan, Qichen Fu, Regan Poston, Sam Xu, Shuangning Liu, Sushma Rao, Tashweena Heeramun, Thomas Merth, Uday Rayala, Victor Cui, Vivek Rangarajan Sridhar, Wencong Zhang, Wenqi Zhang, Wentao Wu, Xingyu Zhou, Xinwen Liu, Yang Zhao, Yin Xia, Zhile Ren, and Zhongzheng Ren. Apple intelligence foundation language models. *ArXiv*, abs/2407.21075, 2024. URL https://api.semanticscholar.org/CorpusID:271546289.

Dengke Han. Draftexpert: Expansion-aware self-speculative decoding for end-device moe inference, 2026. URL https://arxiv.org/abs/2607.24434.

Bairu Hou, Qibin Chen, Jianyu Wang, Guoli Yin, Chong Wang, Nan Du, Ruoming Pang, Shiyu Chang, and Tao Lei. Instruction-following pruning for large language models, 2025. URL https://arxiv.org/abs/2501.02086.

Haochen Huang, Shengxuan Qiu, and Meng Li. S2-moe: Enabling efficient self-speculative decoding for mixture-of-experts on edge devices, 2026. URL https://arxiv.org/abs/2608.15018.

Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. Moesd: Unveil speculative decoding's potential for accelerating sparse moe, 2025. URL https://arxiv.org/abs/2505.19645. NeurIPS 2025 (spotlight).

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991. URL https://api.semanticscholar.org/CorpusID:572361.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts, 2024. URL https://arxiv.org/abs/2402.07871.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:254096365.

Shuang Liang, Hao Mark Chen, Zhiwen Mo, Qianzhou Wang, Guoyu Li, Lingxiao Ma, and Wayne Luk. Acceptmoe: Commitment-weighted self-sizing verifier expert sets for efficient moe speculative decoding, 2026. URL https://arxiv.org/abs/2608.02989.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, and Sebastian Jaszczur. Joint moe scaling laws: Mixture of experts can be memory efficient, 2025. URL https://arxiv.org/abs/2502.05172.

Bradley McDanel, Steven Li, Sruthikesh Surineni, and Harshit Khaitan. Moe-spec: Expert budgeting for efficient speculative decoding, 2026. URL https://arxiv.org/abs/2602.16052.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Daniel Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hanna Hajishirzi. Olmoe: Open mixture-of-experts language models. *ArXiv*, abs/2409.02060, 2024. URL https://api.semanticscholar.org/CorpusID:272366674.

NVIDIA. Nemotron 3 super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning, 2026. URL https://arxiv.org/abs/2604.12374.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016. URL https://arxiv.org/abs/1606.06031.

Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. *ArXiv*, abs/2501.11873, 2025. URL https://api.semanticscholar.org/CorpusID:275787957.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019. URL https://arxiv.org/abs/1907.10641.

Anish Saxena, Po-An Tsai, Hritvik Taneja, Aamer Jaleel, and Moinuddin Qureshi. Utility-driven speculative decoding for mixture-of-experts, 2025. URL https://arxiv.org/abs/2506.20675.

Noam M. Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ArXiv*, abs/1701.06538, 2017. URL https://api.semanticscholar.org/CorpusID:12462234.

Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart van Baalen, Markus Nagel, Paul N. Whatmough, and Babak Ehteshami Bejnordi. Mixture of cache-conditional experts for efficient mobile device inference. *ArXiv*, abs/2412.00099, 2024. URL https://api.semanticscholar.org/CorpusID:274436794.

Shriniwas Ramesh Suram. Cacheable by design? training mixture-of-experts routers for locality against the edge memory-bandwidth wall, 2026. URL https://arxiv.org/abs/2608.18261.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. URL https://api.semanticscholar.org/CorpusID:13756489.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017. URL https://arxiv.org/abs/1707.06209.

Zhenhe Wu, Yaping Jin, Qinghua Xing, Hang Zhou, Wei He, Xianjie Wu, Xianfu Cheng, Jian Yang, and Hanting Chen. Cache-aware joint router adaptation for memory-efficient moe inference, 2026. URL https://arxiv.org/abs/2609.04895.

Xiaomi LLM-Core Team. Mimo-v2-flash technical report, 2026. URL https://arxiv.org/abs/2601.02780.

Jincheng Xie, Runheng Liu, Heyan Huang, Yawen Ling, Hanbin Dai, Yu Zheng, and Wen Hu. Less experts, faster decoding: Cost-aware speculative decoding for mixture-of-experts, 2026. URL https://arxiv.org/abs/2607.12696.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019. URL https://arxiv.org/abs/1905.07830.

Yan-Quan Zhou, Tao Lei, Han-Chu Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. *ArXiv*, abs/2202.09368, 2022. URL https://api.semanticscholar.org/CorpusID:247011948.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *ArXiv*, abs/2202.08906, 2022. URL https://arxiv.org/abs/2202.08906.

A Appendix

B Hyperparameter Settings

We train all models (dense and MoE) with the following settings.

Training. Batch size 512 and sequence length 4096. All models are trained on NVIDIA H100 GPUs.

Optimization. We use the AdaStar optimizer (EMA $= 0.95$) with a learning rate of 0.01, linear warmup for 5000 steps followed by cosine decay ($\alpha = 0.005$), and weight decay 3.16×10^{-4}. Gradients are rescaled to norm 1 if their norm exceeds 1, and updates are skipped entirely if the norm exceeds 100.

Architecture. Vocabulary size $49{,}152$, 16 attention heads, and 8 KV heads.

Loss functions. The dense model loss is cross-entropy plus a z-loss (scale 10^{-6}). The MoE loss additionally includes a load-balancing loss (weight 0.01) and a router z-loss (weight 10^{-4}).